Cover Page

ÖREBRO UNIVERSITET

INSTITUTIONEN FÖR
NATURVETENSKAP OCH TEKNIK

Kursens namn	**Objektorienterad programmering för civilingenjörer–DT506G**
Examinationsmomentets namn/provkod	**A001**
Datum	2022-03-16
Tid	Kl. 08:15 – 13:15

Tillåtna hjälpmedel	
Instruktion	
Viktigt att tänka på	
Ansvarig/-a lärare (ev. telefonnummer)	Farhang Nemati Tel: 019-303095 Mobil: 0702533418
Totalt antal poäng	40
Betyg (ev. ECTS)	The exam contains 9 questions. The total points are 40. At least 24 points are required for grade 3 (pass), 30 points for grade 4 and 35 points for grade 5.
Tentamensresultat	The results will be notified in Studentforum within 15 working days after the exam.
Övrigt	You can write your answers in English or Swedish.

Good Luck!

Questions

1. (3 points)

Consider the class MyClass:

```
class MyClass
{
private:
  int r1;
protected:
  int t1;
public:
  int b1;
  int getB1() { return b1; }
  int getT1() { return t1; }
  int getR1(){ return r1; }
};
```

In the following main() which statements are ok and which one are error. Select OK or Error accordingly. Each statement is identified by the letter in the comment following the statement.

```
int main()
{
  MyClass mc;
  mc.r1 = 100;//a
  int i = mc.getR1();//b
  int j = mc.getB1();//c
  mc.t1 = 5;//d
  mc.b1 = 7;//e
  j = mc.getT1();//f
};
```

	OK	Error

		OK	Error
A	a	○	○
B	b	○	○
C	c	○	○
D	d	○	○
E	e	○	○
F	f	○	○

2. (4 points)

Write a class (name it *MyST*) that has two data members:

a. *totalObj*: keeps the total number of objects that have been instantiated from the class since the start of the program.

b. *currentObj*: keeps the number of objects of the class that currently exist. It means it holds the number of objects that have been constructed but not destructed yet.

c. What will be the out put of the following code?

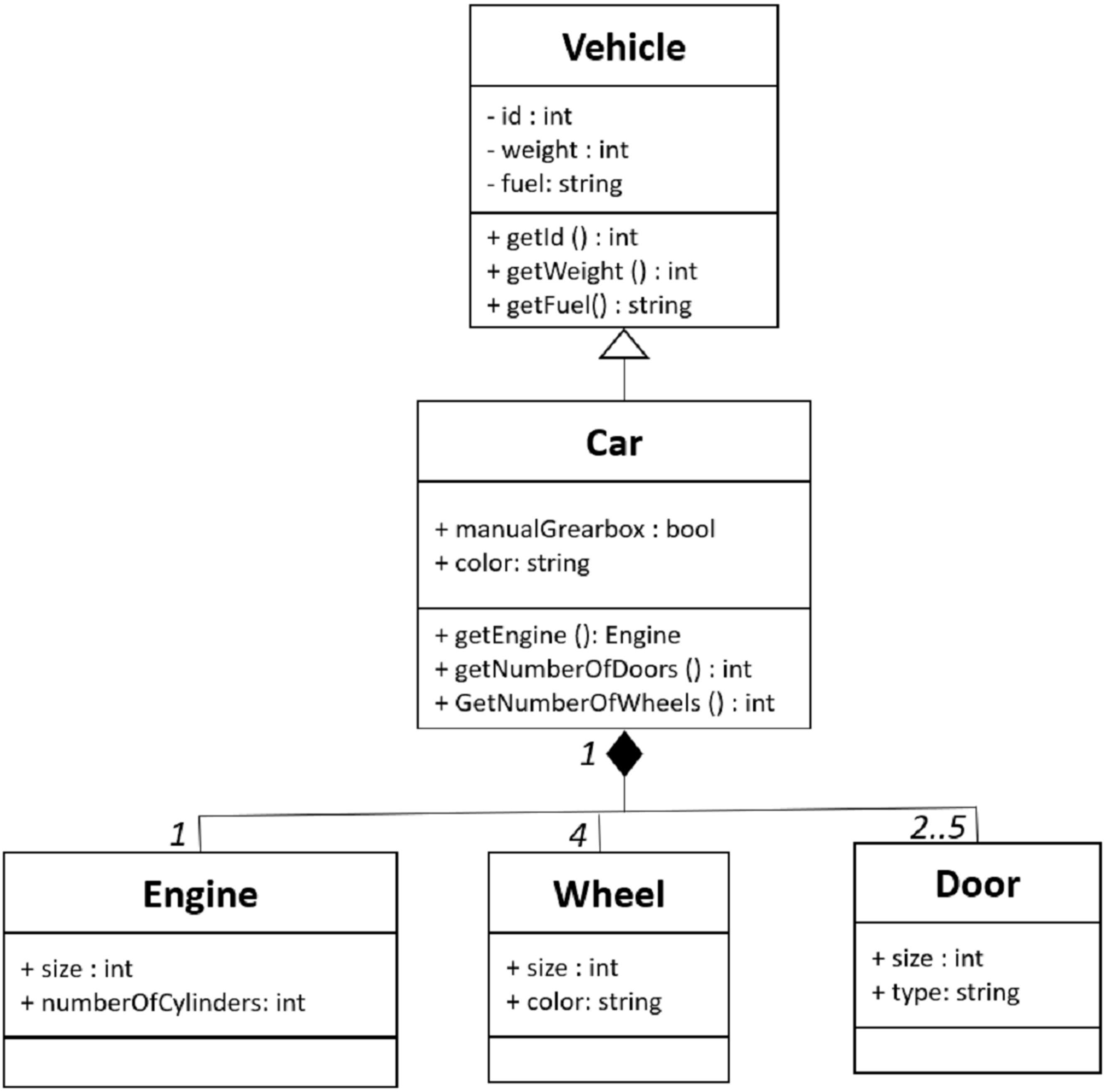
```
using namespace std;

int main()

{

  MyST m1;

  cout << m1.totalObj << " " << m1.currentObj << endl;

  MyST m2;

  cout << m2.totalObj << " " << m2.currentObj << endl;

  {

    MyST m3;

    cout << m3.totalObj << " " << m3.currentObj << endl;

  }

  MyST m4;

  cout << m4.totalObj << " " << m4.currentObj << endl;

};
```

B *I* U | ☰ ☰ á

0 / 10000 Word Limit

3. (8 points)

Define the classes that are in the following class diagram. Notice that a minus symbol ('-') before the class member means it is private and a plus symbol ('+') means that the member is public.



B *I* U

0 / 10000 Word Limit

4. (5 points)

Consider the following class:

```
class MyClass
{
private:
  int m_a;
 double m_b;
public:
  MyClass(): m_a(0), m_b(0)
  {}
  MyClass(int a, double b): m_a(a), m_b(b)
  {}
  // Add the overloaded operators here
};
```

a. Overload operator **+** that adds two objects of MyClass. It must return an object where its m_a and m_b are respectively the summation of m_a and m_b of the two given objects.

b. Overload operator **==** that compares two objects of MyClass. If both m_a and m_b of the two objects are equal it should return true otherwise it returns false.

c. Overload operator **()** that prints out the values m_a and m_b of the object of MyClass.

Example:

```
using namespace std;
int main()
{
  MyClass c1(2, 3.1);
 MyClass c2(8, 4.2);
 MyClass c3 = c1 + c2;
  cout << c3() << endl;
  if(c1 == c2)
    cout << "c1 is equal to c2" << endl;
  else
    cout << "c1 is NOT equal to c2" << endl;
};
```

Output:

```
m_a: 10  m_b: 7.3

c1 is NOT equal to c2
```

0 / 10000 Word Limit

5. (3 points)

To the definition of class MyArray add copy and assignment constructors so that copying and assignments are done with deep copy and assignment .

```
class MyArray
{
private:
  int* m_array;
  int m_size;
public:
  MyArray(): m_size(0), m_array(nullptr) {}
  MyArray(int size): m_size(size)
  {
   if(m_size >= 0)
        m_array = new int[m_size];
   else
        m_size = 0;
  }
 ~MyArray()
 {
   delete[] m_array;
  }
  int& operator[](int index)
  {
    if(index < m_size && index >= 0)
         return m_array[index];
  }
};
```

B I U

0 / 10000 Word Limit

6. (6 points)

Consider the following classes. What would be the output of the code in the following main()?

```
class Base
{
public:
  virtual std::string who() { return "Base"; }
};
class D1: public Base
{
public:
  std::string who() { return "D1"; }
};
class D2: public D1
{
public:
  std::string who() { return "D2"; }
};
class D3: public D2
{
public:
  std::string who() { return "D3"; }
};

using namespace std;
int main()
{
  D1 d1;
  D2 d2;
  D3 d3;
  Base b;
  Base* bPtr;
  bPtr = &b;
  std::cout << bPtr->who() << std::endl;//a
  bPtr = &d3;
  std::cout << bPtr->who() << std::endl;//b

  bPtr = &d2;

  std::cout << bPtr->who() << std::endl;//c

  bPtr = &d1;

  std::cout << bPtr->who() << std::endl;//d

  D1* dd = new D2();

  std::cout << dd->who() << std::endl;//e

  delete dd;

  dd = new D3();

  std::cout << dd->who() << std::endl;//f

  delete dd;

}
```

0 / 10000 Word Limit

7. (5 points)

a. Define a class called Dog that implements both following interfaces.

b. Define a class called Car that implements interface `OnlineStoreItem` only.

What extra data members your Car and Dog classes will have, is not important here.

c. Define one global function (name it *printPrice*) that prints out the price of the object passed to it. It should take any object of any class that implements interface `OnlineStoreItem`.

d. In a main() function declare one (only one) vector to which you add pointers to a few objects of both classes Dog and Car. Notice that you have to add the pointers of all the objects of both classes to the same vector (not two separate vectors). Then iterate through your vector and print the price of all the objects using your function *printPrice*.

```
class Animal

{

public:

  virtual void speak() = 0;

  virtual std::string getName() = 0;

};

class OnlineStoreItem

{

public:

  virtual double getPrice() = 0;

  virtual void setPrice(double price) = 0;

};
```

B *I* U á

0 / 10000 Word Limit

8. (4 points)

a. Define a template function that returns the maximum value of its given two arguments:

```
type max(type t1, type t2)
```

b. Define a specialization of your template function for type Person (defined below). The specialized template function should return the person who is older.

```
class Person

{

public:

  std::string m_name;

  int m_age;

};
```

B *I* U | ☰ ☰ á

0 / 10000 Word Limit

9. (2 points)

Briefly (in one or two sentences) explain the following design principles:

a. The Interface Segregation Principle (ISP)

b. The Open-Closed Principle (OCP).

B *I* U | ☰ ☰ á

0 / 10000 Word Limit